CS 4476: Computer Vision, Fall 2020

PS1

Instructor: Devi Parikh

Due: Thursday, September 10th, 11:59:00 pm

Instructions

1. The deliverables for this assignment must be submitted on Gradescope. Please follow the submission instructions carefully.
   - Save your written answers to a pdf file. LaTeX is strongly encouraged.
   - Answers to each problem must be on a separate page (or pages). On Gradescope uniquely assign each page to a single problem. Multiple sub-problems can be included on the same page.
   - Do not include your name on the answer sheet.
   - Upload your pdf to the `PS1` assignment on Gradescope.
   - Save your code and output images in a zip file.
   - Be sure to include all the files listed in the submission checklist.
   - Upload your zip file to the `PS1 Code` assignment on Gradescope.

Getting Started

In this assignment you will write code in a Jupyter notebook. If you are new to Jupyter notebooks, we have put together this tutorial.ipynb file. To run the tutorial, install the Jupyter Notebook Interface and then read the docs to figure out how to view the `.ipynb` file.

1 Short answer problems [30 points]

1. Give an example of how one can exploit the associative property of convolution to more efficiently filter an image.
2. This is the input image: `[0 0 1 1 0 0 1 1]`. What is the result of dilation with a structuring element `[1 1 1]`?
3. The filter f' = `[0 -1/2 0 1/2 0]` is used as the filter to compute an estimate of the first derivative of the image in the x direction. What is the corresponding second derivative filter `f"`. (*Hint:* Assymetric filters must be flipped prior to convolution.)
4. Name two specific ways in which one could reduce the amount of fine, detailed edges that are detected with the Canny edge detector.
5. Describe a possible flaw in the use of additive Gaussian noise to represent image noise.

6. Design a method that takes video data from a camera perched above a conveyor belt at an automotive equipment manufacturer, and reports any flaws in the assembly of a part. Your response should be a list of concise, specific steps, and should incorporate several techniques covered in class thus far. Specify any important assumptions your method makes.

2 Programming problem: content-aware image resizing [70 points]

For this exercise, you will implement a version of the content-aware image resizing technique described in Shai Avidan and Ariel Shamir's SIGGRAPH 2007 paper, "Seam Carving for Content-Aware Image Resizing". The goal is to implement the method, and then examine and explain its performance on different kinds of input images.

First read through the paper, with emphasis on sections 3, 4.1, and 4.3. Note: choosing the next pixel to add one at a time in a greedy manner will give sub-optimal seams; the dynamic programming solution ensures the best seam (constrained by 8-connectedness) is computed. Use the dynamic programming solution as given in the paper and explained in class.

Part A: Implementation

Follow the instructions in `SeamCarving.ipynb` to implement the functions below. `SeamCarving.ipynb` is included in this zip file. After you are done use the `notebook2script.py` script to export a file called `submissionSeamCarving.py`. Call the functions in this file to complete Part B of this assignment. Include this file in your code submission.

In the notebook you will implement:

- `gx, gy = compute_gradients(img)` - to compute the x and y gradients of the image. The input `img` will be a `MxNx3` matrix of datatype `uint8`. The output should be two 2D matrices of data type `float64`.
- `energyImage = energy_image(img)` - to compute the energy at each pixel using the magnitude of the x and y gradients (equation 1 in the paper). The input `img` will be a `MxNx3` matrix of datatype `uint8`. The output should be a 2D matrix of dataype `float64`.
- `cumulativeEnergyMap = cumulative_minimum_energy_map(energyImage, seamDirection)` - to compute minimum cumulative energy. The input `energyImage` will be a 2D matrix of datatype `float64`. (It can be the output of `energy_image` function defined above.). The input `seamDirection` will be the string '`HORIZONTAL`' or '`VERTICAL`'. The output should be a 2D matrix of datatype `float64`.
- `verticalSeam = find_optimal_vertical_seam(cumulativeEnergyMap)` - to compute the optimal vertical seam. The input will be a 2D matrix of datatype `float64`. (It can be taken from the output of the `cumulative_minimum_energy_map` function defined above). The output should be a vector containing the column indices of the pixels which form the seam for each row.
- `horizontalSeam = find_optimal_horizontal_seam(cumulativeEnergyMap)` - to compute the optimal horizontal seam. The input will be a 2D matrix of datatype `float64`. (It can be taken from the output of the `cumulative_minimum_energy_map` function defined above). The output should be a vector containing the row indices of the pixels which form the seam for each column.
- `display_seam(img, seam, seamType)` - to display the selected type of seam on top of an image. The color input `img` for which the seam was computed. The `seam` can be the output of `find_optimal_vertical_seam` or `find_optimal_horizontal_seam`. The `seamType` can be the strings '`HORIZONTAL`' or '`VERTICAL`'. The function should display the input image and plot the seam on top of it.

- Functions with the following interface:
  `reducedColorImage, reducedEnergyImage = reduce_width(img, energyImage)`
  `reducedColorImage, reducedEnergyImage = reduce_height(img, energyImage)`
  These functions should take as inputs `a)` a `MxNx3` matrix `img` of datatype uint8 and `b)` a 2D matrix `energyImage` of datatype `float64`. The input `energyImage` can be the output of the `energy_image` function. The output should return 2 variables: `a)` a 3D matrix same as the input image but with its width or height reduced by one pixel; `b)` a 2D matrix of datatype `float64` same as the input `energyImage`, but with its width or height reduced by one pixel.

- Functions with the following interface:
  `reducedColorImage = seam_carving_reduce_width(img, reduceBy)`
  `reducedColorImage = seam_carving_reduce_height(img, reduceBy)`
  These functions take as inputs a) a `MxNx3` matrix `img` of datatype uint8 and b) a integer specifying the number of pixels to reduce the input image by along its width or height. The output should be a 3D matrix same as the input image by with its width or height reduced by the appropriate amount.

Part B: Experimentation

Answer each of the following as indicated:

1. For each of the input images `inputSeamCarvingPrague.jpg` and `inputSeamCarvingMall.jpg` (included in the zip file from Part A), display in your answer sheet the result of reducing the **width** of the image by 100 pixels using the `seam_carving_reduce_width` function. (Part of this was already completed in Part A.)

2. For each of the input images `inputSeamCarvingPrague.jpg` and `inputSeamCarvingMall.jpg` (included in the zip file from Part A), display in your answer sheet the result of reducing the **height** of the image by 100 pixels using the `seam_carving_reduce_height` function. (Part of this was already completed in Part A.)

3. For the input image `inputSeamCarvingPrague.jpg`, display in your answer sheet: (a) the energy function output, and (b) the two corresponding cumulative minimum energy maps for the seams in each direction (use `matplotlib.pyplot.imshow`). (c) Explain why these outputs look the way they do given the original image's content.

4. For the same image `inputSeamCarvingPrague.jpg`, display in your answer sheet: the original image together with (a) the first selected horizontal seam and (b) the first selected vertical seam in your answer sheet. (c) Explain why these are the optimal seams for this image.

5. Make some change to the way the energy function is computed (i.e., filter used, its parameters, or incorporating some other prior knowledge). In your answer sheet: (a) explain the changes, (b) display an example result, and (c) explain the impact of the changes with reference to the example. Please do not submit this code.

6. Now, for the real results! Use your system with different kinds of images and seam combinations, and see what kind of interesting results it can produce. The goal is to form some perceptually pleasing outputs where the resizing better preserves content than a blind resizing would, as well as some examples where the output looks unrealistic or has artifacts.

   Include results for at least three images of your own choosing. Include an example or two of a "bad" outcome. Be creative in the images you choose, and in the amount of combined vertical and horizontal carvings you apply. Try to predict types of images where you might see something interesting happen. It's ok to fiddle with the parameters (seam sequence, number of seams, etc) to look for interesting and explainable outcomes.

For each result, include the following things on your answer sheet (clearly labeled):

(a) the original input image.

(b) your system's resized image,

(c) the result one would get if instead a simple resampling were used (e.g. via `scipy.misc.imresize`)

(d) the input and output image dimensions,

(e) the sequence of removals that were used

(f) a qualitative explanation of what we're seeing in the output.

3 [OPTIONAL] Extra credit [up to 10 points each, max possible 20 points extra credit]

Below are ways to expand on the system you built above. If you choose to do any of these (or design your own extension) include in your answer sheet, an explanation of the extension as well as images displaying the results and a short explanation of the outcomes. Also include a line or two of instructions telling us what needs to be done to execute the extra credit portion of your code.

1. Implement functions to increase the width or height of the input image, blending the neighboring pixels along a seam. (See the Seam Carving paper for details.) Add an example to your answer sheet that clearly demonstrates the impact.

2. Implement the greedy solution and compare the results to the optimal Dynamic Programming solution. In your answer sheet: provide pseudocode, an example result, and explain the differences with reference to the example.

Python hints:

1. Useful modules: `numpy`, `scipy`, `matplotlib`, `imageio`

2. `scipy` has different convolution functions: `scipy.ndimage.filters` and `scipy.signal.convolve`, see the difference and choose the appropriate one.

3. Be careful with `double` and `uint8` conversions as you go between computations with the images and displaying them.

Submission Checklist

☐ Answer sheet

☐ `Zip` file containing:

  ☐ `submissionSeamCarving.py`

---

1. This assignment is adapted from the PS1 assignment from Kristen Grauman's CS 376: Computer Vision at UT Austin.
2. Image acknowledgements: Thanks to the following Flickr users for sharing their photos under the Creative Commons license: inputSeamCarvingMall.jpg is provided by hey tiffany! inputSeamCarvingPrague.jpg david.nikonvscanon.
3. Be sure to credit any photo sources.